STATE ESTIMATION FOR K9

Ru-Gang Xu
NASA Ames Research Center
July 31, 2001

Reviewed by NASA-USRP Mentors
Mike Fair, Maria Bualat
Code IC

Abstract

This paper introduces the estimator module, appropriately named as "Estimate", for the K9 Mars Rover. The estimator produces both attitude and position by Kalman Filtering gyro (IMU), compass, inclinometer, and encoder data. It is designed to combine asychronous data from a wide range of sensors making it robust and flexible. It is robust in the sense that if any of devices stop sending information, it will continue to estimate state with the current working sensors. It is flexible, because it can accept data from any device as long as the device specifics are known. This paper describes the motivation, the specifics, and the results of this estimator.

1 Introduction:

The motivation of this project is to produce robot better state information while limiting the sensors used to be those applicable on Mars. Currently there is no simple instrument that can give an absolute position on Mars. Therefore, the only positional information available is relative, specifically: dead reckoning. However, classical dead reckoning can be improved with the addition of better attitude information.[1,2]

In the past, the K9 Rover used classical dead reckoning, relying completely on encoder information, for state estimation. From the encoder data, both a yaw and translational delta vector were produced. Both vectors were then transformed into earth frame coordinates by previously calculated attitude information. The vectors were then added with the previously calculated state.

The limits of such a system, especially one being used on uneven terrain, are obvious. Classical dead reckoning assumes three major things: flat terrain, no wheel slippage, and perfect knowledge of wheels. These assumptions are not valid on sandy uneven terrain. As the robot goes up and down a hill, classical dead reckoning will overestimate the true distance traveled. If the wheels slip, movement would be detected even though no movement has occurred. If wheel alignment or size were to be slightly off, a curved path would result while classical dead reckoning believes it is a straight path. As all dead reckoning data are integrated, all errors are integrated resulting in unbound growth of errors.

All three causes of inaccuracy in classical dead reckoning can be attenuated with the addition of attitude information. If the absolute attitude was known, a matrix transforming robot-frame to earth-frame can be accurately produced. This matrix can be applied to the translational data produced by dead reckoning. This eliminates the overestimate given during hills, as the inclinations are known. This also eliminates any false yaw information given by wheel slippage. Perfect wheel knowledge is also not needed as if the wheels are misaligned or of different sizes, knowledge of orientation would produce the correct curved path. Although some translational errors cannot be helped (i.e. translational motion resulting from slippage), the addition of accurate attitude eliminates many other errors.

2 *Estimate*:

The goal of *Estimate* is to take advantage of attitude information to produce better pose while staying flexible and robust.

Currently there are several instruments that are used for attitude: gyros, inclinometers, and compasses. However, precise and useful attitude information cannot come from one instrument. Integration of rotational rates, from gyro data for example, would result in drift. Therefore, although gyros are accurate in the short-term, accuracy in the long term is unlikely. Using absolute instruments such as compass and inclinometers can result in an accurate measurement of attitude in the long term. However, in the short term, the physical nature of compasses and inclinometers, and the dynamic nature of a mobile platform result in highly volatile and therefore useless data.

The solution then is to use both absolute and relative data. Kalman Filtering is known to be able to combine gyro and compass/inclinometer data to produce stable and accurate attitude information. Since the model of motion is linear and the data comes in as discrete samples, a Discrete Kalman Filter was selected as the core of the new estimator. Therefore, *Estimate* can be divided into two parts: the Discrete Kalman Filter and the code framework.

2.1 The Discrete Kalman Filter:

2.1.1 Introduction to Discrete Kalman Filtering

Kalman Filtering has been a well-researched topic in the past forty or so years proving itself as an excellent choice for navigational sensor fusion. Although there are several variations in the technique, the one of primary interest in navigation is the Discrete Kalman Filter.

A Discrete Kalman Filter is described as an *optimal recursive linear estimator*.[4] In other words, a Kalman Filter estimates the state of a noisy linear system (x_k) by using a previous estimate and any new measurements (z_k). The system is noisy in the sense that there is both noise in the system described (w_{k-1}) and noise in the measurement (v_k):

$$x_k = Ax_{k-1} + Bu_k + w_{k-1} (1)$$

$$z_k = Hx_k + v_k (2)$$

Equation (1) is the linear stochastic difference equation describing the system, while equation (2) describes the measurement z_k with respect to x_k. x_k is a column vector of size n, and z_k is a column vector of size m. Then the $n \times n$ matrix A predicts current state from the previous state.

The $n \times l$ matrix B then relates the optional control input u, a column vector of size l, to the state. The $m \times n$ matrix H relates the state to the measurement.

In order to make a Kalman Filter possible in the sense of making the mathematics tractable and the solution well defined, the noise must be white Gaussian noise. Whiteness implies noise not correlated with time (i.e. equal power in all frequencies), while Gaussian implies that the noise is described by a Gaussian probability density curve centered at zero (i.e. $p(w) \sim N(0, Q)$ and $p(v) \sim N(0, R)$ where Q is the process noise covariance matrix and R is the measurement noise covariance matrix).

Therefore, the state of such a system can only be a vector of Guassian distributions. If given all the variances and means of the system and measuring devices, the measured input and the previous state, a Kalman filter can produce a vector of the means and variances describing the current state. There is no information lost since a Gaussian distribution can be completely described by its first and second order statistics (i.e. mean ($\hat{x}_k$) and covariance (P_k)). Therefore, the new statistics can then be reentered into the Kalman Filter for the next estimate. These statistics are derived by two different equations:

$$\hat{x}_k = \hat{x}_k^- + K_k(z - H\hat{x}_k^-) \quad (3)$$

$$P_k = (I - K_k H)(P_k^-) \quad (4)$$

where $P_k^- = AP_{k-1}A^T + Q$ and $\hat{x}_k^-$ is the derived mean of x_k from equation (1).

The principle component of those two equations is the Kalman gain K_k. K_k is picked to minimize the covariance P_k (i.e. equation 4), therefore:

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$$

In summary, the Discrete Kalman Filter can be implemented in 5 simple equations, divided as the time update and the measurement update equations:[3]

time update:

$$\hat{x}_k^- = A\hat{x}_{k-1}$$

$$P_k^- = AP_{k-1}A^T + Q$$

measurement update:

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$$
$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$
$$P_k = (I - K_k H) P_k^-$$

The time update stage can be thought of as the prediction stage where the next estimate is predicted from the previous values. The measurement update stage can be thought of as the correction stage where the predicted values are corrected by the new measurements. This leads to a simple implementation in code.

2.1.2 The Function *Kalman*

Kalman Filtering in *Estimate* is used to combine data from any instrument for pose estimation. The flexibility of the Discrete Kalman Filter and the similarities between position and attitude allow the creation of a simple single function, called "Kalman", that is applicable to all types of pertinent data (distance, velocity, acceleration). The measurement equation (2) allows specification of the type of data entered, by H. This allows any type of device to input information as long as the type of information is known. Also, both position and attitude have the same linear model, which can be divided into each axis (x,y, and z). Therefore, without losing any generality, *Kalman* can be designed for simple one-dimensional linear motion. This eliminates the use of large sparse matrices while creating a more flexible function.

Kalman is basically just the five Discrete Kalman Filter equations applied to one dimensional linear motion where the state x_k is a vector of distance, velocity and acceleration with no input u, so:

$$A = \begin{pmatrix} 1 & \triangle T & (\triangle T)^2/2 \\ 0 & 1 & \triangle T \\ 0 & 0 & 1 \end{pmatrix}$$
$$B = 0$$

The inputs of the function are H, P_{k-1}, x_{k-1}, z_k, $\triangle t$, and R, and the output is the next estimate (x_k, P_k). The inputs allow the function to be applied to all axes and from all instruments. The only thing that remains is setting up the many calls to *Kalman*.

2.2 The Framework

The rest of the code involves using *Kalman* to input sensor data to a central repository for pose.

All estimated information is stored in one single struct, the *statetype* The *statetype* includes the pose, velocities, and accelerations, the calculated covariances, and last time of update.

This struct is accessed through four functions: *estimate_init*, *estimate_get_pos*, *estimate_get_ attitude*, and *estimate_update*. *Estimate_init* initializes the repository. *Estimate_get_pose* and *estimate_get_location* gets pose and location respectively. *Estimate_update* updates the repository.

Estimate_get_pose and *estimate_get_location* can be called anytime. It simply copies the pose and location from the repository.

Estimate_update is called after a measurement has been made. An instrument calls *estimate_update* with its ID, data, and time of measurement. This new data is then Kalman Filtered with the repository data. This function is the bulk of *Estimate*.

When *estimate_update* is called, a different path is taken depending on the device ID, however, the basic idea is the same: copy the current repository, convert data into earth frame, load the various inputs for *Kalman*, and copy the *Kalman* outputs into the repository. Conversion of the input data is done through a transformation matrix created from previously calculated attitude. The complete inputs needed by *Kalman* are supplied by the repository and the device. The repository gives the previous covariance and mean while each device is described by a *Device* struct. *Device* contains the measurement variance, the type of measurement (H and motion type), the raw data from the device, the time of last update, whether a device is in use, and if the device has given data yet. The *Device* gives H and the measurement variance (R) to *Kalman*. With this information, *Kalman* can be called once for each axis to predict the complete state. The new estimate is then copied into the repository.

This scheme allows *Estimate* to work in an asynchronous environment. If a device does not call estimate_update due to failure, congestion, etc., *Estimate* will only lose some accuracy during that period. This scheme also allows *Estimate* to be easily modified to accommodate new instruments. To add a new instrument, a new Device object and a new case of device ID must be created. The new case would just be simply three *Kalman* calls, one each for each axis, using earth-frame data, the previous estimate, and the Device as inputs.

3 *Estimate* Applied

3.1 *Estimate* on K9

Estimate is designed to be used on the K9 rover. Therefore, many initialization parameters were needed, specifically the variances of the various instruments and the process noise.

K9 currently contains an IMU[5], a TCM[6], and encoders. The IMU is a package of three gyros and three accelerometers(one each for each axis). The TCM gives absolute roll (-50° to 50°), pitch (-50° to 50°), and yaw (0° to 360°) through a magnetic compass and inclinometers. The encoders go through a dead-reckoning function to produce rover frame translational and yaw deltas.

All three instruments were never accurately measured for covariances, and an accurate assessment of the process noise was never made due to time constraints. Instead, data was gathered from the IMU and TCM as K9 was moving. Only the gyros of the IMU were used. The accelerometers' drift rates changed depending on how K9 was moving, therefore, it was omitted from the filter. The remaining sensors' filter values were tweaked in Matlab so that Kalman Filtered values came close to real and useful values. Through trial and error, a well-filtered *Estimate* was produced.

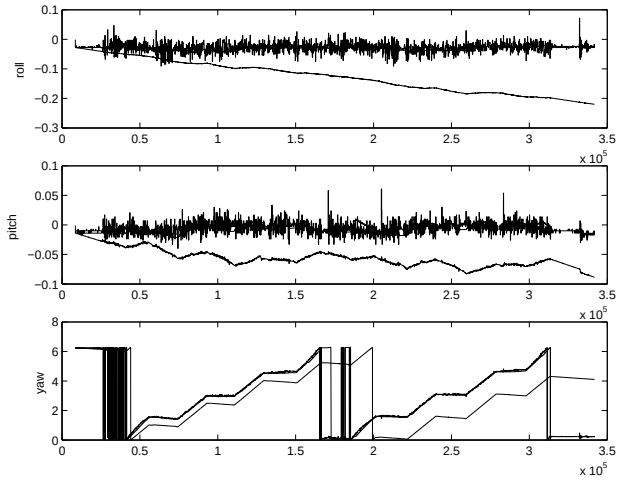

Figure 1: TCM(noisy signal), IMU(drifting signal), Filtered(clean signal following the TCM)

The noisy TCM data fused with the drifting IMU data produced a relatively stable and accurate output.

3.2 Results on K9

After *Estimate* was decently calibrated and *Estimate* integrated with the navigation and control software, K9's navigational abilities were tested both indoors and in the field.

A simple test of rotation was done indoors. 360° turns were made with just dead reckoning and with full use of estimate. Using just encoders, turns resulted in an error on the average around 30°. *Estimate* turns produced slight differences (5°-10°), but these were not errors of estimation, but rather of control. With *Estimate*, K9 knew it overshot or undershot its target, but the motors had no feedback.

The other inside run was to create .5 m by .5 m squares on a cement floor by issuing direct commands rather than using the navigation software. Four types of runs were made: just dead reckoning, dead reckoning with IMU and compass, dead reckoning with IMU, and dead reckoning with compass. The last two served as tests that involved failure of devices. Each run was compared to one of another type because comparison between a real square would be unreasonable as again there is no feedback in direct motor control. K9 without the compass produced totally degenerate squares after the first square was made. K9 with the compass produced reasonable squares, however, due to wheel slippage, the square would shift in a direction depending whether it was a counterclockwise or clockwise square.

The field tests were more impressive. K9 ran two types of field runs. The first type was on relatively flat terrain. The path was a 4 m by 4 m square made by the navigation software (i.e. start at (0,0), make a square and return to (0,0)). In this case, an absolute comparison can be made as the path created resulted from feedback of information gathered by *Estimate*. The difference between the starting point and the final position of K9 is the error. The percent error can be calculated by dividing the magnitude of the error by the total distance traveled. After driving one square:

	Error in (x,y)	Odometer	% Error
Encoder	(3.7 m, 1.17 m)	16.2 m	24
Encoder, IMU, TCM	(0 m, 0.9 m)	15.2 m	5.9

The second type of test was a path which included a hill (3 meters long and .5 meters high). Using the navigation software, a path across the hill and back to the starting point was made. This was the most interesting test as the path was very uneven and the slippage occurred many times:

	Error in (x,y)	Odometer	% Error
Encoder	(4.63 m, 1.7 m)	14 m	35
Encoder, IMU, TCM	(.25 m, -1.37 m)	17 m	8

4 Conclusion

Although few field tests were made due to time constraints, the positive impact of using *Estimate* is apparent, however room for improvement can always be made.

The tests done were not complete to pinpoint future improvements. Future tests should be run with measurements of ground truth perhaps with GPS. With a complete comparison of the path traveled, hints on causes of errors can be found.

Hopefully, fully calibrating *Estimate* will result in the elimation of some of those errors. *Estimate* currently uses constant measurement and process covariance matrices. With these matrices

as contants, Kalman Filter covariance converges, making *Estimate* become an implementation of recursive weighted squares. On K9, the true values of those matrices are hardly constant. Dynamic modeling of K9 could be used to produce those covariances. The more complete solution would be to use Particle Filtering to model each sensor as K9 is moving. Another solution would be to use a Kalman Smoother on raw telemetry data from all the sensors to construct a covariance for each type of run.

In conclusion, the haphazardly calibrated *Estimate* works. It would be very interesting to see it applied to other robots. However, future work is needed for better calibration and more field tests.

5 References:

1. Borenstein, J. and Fong, L, 1996, *Gyrodometry: A New Method for Combining Data from Gyros and Odometry in Mobile Robots*, Proceedings of the 1996 IEEE Conference on Robotics and Automation, Minneapolis, Apr 22-28, 1996, pp 423-428

2. Fuke, Y and Krotkov, E., 1996, *Dead Reckoning for a Lunar Rover on Uneven Terrain*, Proceedings of the 1996 IEEE Conference on Robotics and Automation, Minneapolis, Apr 22-28, 1996, pp 411-416

3. Welch G. and Bishop G., 2001, *An Introduction to the Kalman Filter, TR 95-041, University of North Carolina at Chapel Hill*, Chapel Hill, NC

4. Maybeck, P. S., 1979, *Stochastic Models, Estimation, and Control Volume I*, Academic Press, Inc.

5. Inertial Science Inc., 1999, *Inertial Measurement Unit*, Inertial Science Inc, Newbury Park, CA

6. Precision Navigation Inc., 1999, *TCM2 Electronic Compass Module: User's Manual*, Revision 1.08, Precision Navigation Inc., Santa Rosa, CA